

Grammars and Semantics

- Programming languages are used to specify computations – that is, computations are the meaning/semantics of programs.

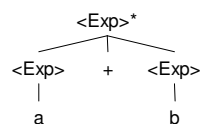
Read Chap 3

Grammars and Semantics

Consider the simple language of expressions:

```
G: <Exp>* ::= <Exp> + <Exp>
      | <Exp> * <Exp>
      | a
      | b
      | c
```

When we write the sentence $a + b$ we can build the parse tree:

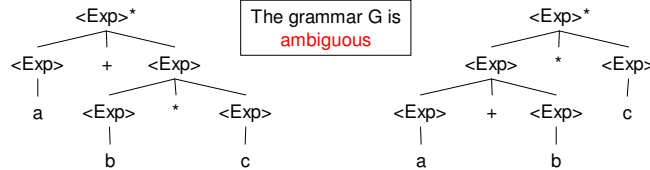


We can say that this parse tree *represents* the computation $a + b$.

If we let a and b be variables, then the parse tree gives us a procedure to compute $a + b$ by starting at the leaves of the tree: (1) lookup the values of the variables (2) pass the values up along the parse tree branches (3) use the values to compute the value of the $+$ operator.

Grammars and Semantics

Now consider the sentence $a + b * c$, for this sentence we can construct two parse trees:



The grammar G is **ambiguous**

Even though both parse trees derive the same terminal string, the computations they represent are very different:

- (1) left tree – first compute the product, then the addition
- (2) right tree – first compute the addition, then the product

Since we had written the original sentence without parentheses the left parse tree represents the intended computation according to algebraic conventions.

However, from a machine point of view, there is no way of knowing which parse tree to pick...

Grammars and Semantics

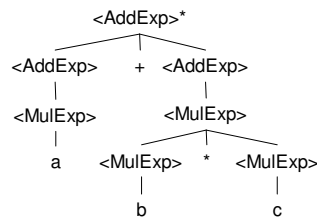
...we need additional information: operator precedence

Operator precedence means that some operators bind tighter than others, e.g. $*$ binds tighter than $+$.

We can build operator precedence right into our grammar:

```
G' : <AddExp> ::= <AddExp> + <AddExp>
      | <MulExp>
    <MulExp> ::= <MulExp> * <MulExp>
      | a | b | c
```

Let's try our problematic sentence $a + b * c$, only one parse tree is possible:



This is the only parse tree we can build, therefore, the grammar G' is not ambiguous.